# EECS 183 Winter 2021

## Exam 2

Q1. Type your **full name** below to affirm the Honor Code pledge: "I have neither given nor received aid on this exam, nor have I concealed any violations of the Honor Code."

Q2. In C++, a member function of a class that has the same name as the class would be described as which of the following?

- (x) Constructor
- ( ) Setter
- ( ) Getter
- ( ) Private member
- ( ) Public member

Q3. Which of the following stream states would evaluate to `false` after attempting and failing to open a file?

- ( ) bad
- ( ) fail
- (x) good
- ( ) eof

Q4. When would we want to use a pass-by-reference parameter for a function? Select all that apply.

- (x) When an argument data type occupies a very large amount of memory.
- ( ) When an argument data type occupies a small amount of memory.
- (x) When we want to alter the original argument.
- ( ) When we do not want to alter the original argument.
- ( ) When a function needs to return at most one value to the calling function.
- (x) When a function needs to modify more than one value in the calling function.

Q5. Consider the following buggy function definition for the next several questions.

```
1.  const int MAX_SIZE = 5;
2.  /**
3.   * Requires: size > 0 and size <= MAX_SIZE
4.   * Modifies: nothing
5.   * Effects:  returns the *greatest* number in row
6.   *
7.   * Example:  If row has the following values
8.   *           3  -20 10
9.   *           for size == 3 the function would return 10
10. */
11. int findGreatest(int row[MAX_SIZE], int size) {
12.     int greatest = 0;
13.     for (int i = 0; i < size; i++) {
14.         greatest = 0;
15.         if (row[i] >= greatest) {
16.             greatest = row[i];
17.         }
18.     }
19.     return greatest;
20. }
```

Q6. 1. Given the following declaration in `main`, Which call(s) to `findGreatest` could expose any of the bugs? Select all that apply.

```
int t1[MAX_SIZE] = { 5, 1, 2, 3, 4 };
```

- ( ) `cout << findGreatest(t1, 1) << endl;`
- (x) `cout << findGreatest(t1, 3) << endl;`

- ( ) `cout << findGreatest(t1, 5) << endl;`
- ( ) None of these tests could expose a bug.

Q5.2. Given the following declaration in `main`, which call(s) to `findGreatest` could expose any of the bugs? Select all that apply.

```
int t2[MAX_SIZE] = { -10, -1, -2, 10, 10 };
```

- ( ) `cout << findGreatest(t2, 1) << endl;`
- ( ) `cout << findGreatest(t2, 2) << endl;`
- ( ) `cout << findGreatest(t2, 4) << endl;`
- ( ) None of these tests could expose a bug.

Q5.3. Which lines of the function have a bug that will cause the function to return a value that is not consistent with the behavior specified in the RME? Select all that apply.

- ( ) 11
- ( ) 12
- ( ) 13
- ( ) 14
- ( ) 15
- ( ) 16

Q5.4. Considering the following test case, select the *arguments* that are considered pass-by-reference. Select all that apply.

```
int t3[MAX_SIZE] = { -10, -1, -2, 10, 10 };
int size = 1;
cout << findGreatest(t3, size) << endl;
```

- ( ) `t3`
- ( ) `size`
- ( ) `MAX_SIZE`

Q6. Given the following code snippet, select a conditional statement to determine if the file **failed** to open.

```
ifstream ins;
ins.open("fileIO.txt");
// choose an option below to fill in the following blank
_____ {
    cout << "File failed to open!";
}
```

- ( ) `if (ins.good()) {`
- ( ) `if (ins.fail()) {`
- ( ) `if (ins.eof()) {`

Q7. Which parameters for `foo` would make the output of the program `35`?

```
void foo(/* which parameters go here */) {
    x += 2;
    y += x;
    return;
}
int main() {
    int a = 3;
    int b = 0;
    foo(a, b);
    cout << a << b;
    return 0;
}
```

- ( ) `int x, int y`
- ( ) `int& x, int& y`
- ( ) `int& x, int y`

- ( ) `int x, int& y`
- ( ) None of the above.

Q8. Refer to the following class definition for the next several questions.

```
class Vegetable {
public:
    /**
     * Requires: Nothing.
     * Modifies: name, weight
     * Effects:  Default constructor. Sets weight to 0,
     *           name to ""
     */
    Vegetable();
    /**
     * Requires: Nothing.
     * Modifies: name, weight
     * Effects:  Non-default constructor. Sets weight to 0,
     *           name to nameIn
     */
    Vegetable(string nameIn);
    /**
     * Requires: Nothing.
     * Modifies: Nothing.
     * Effects:  returns name
     */
    string getName();
    /**
     * Requires: Nothing.
     * Modifies: weight
     * Effects:  increment weight by 1
     */
    void grow();
    /**
     * Requires: Nothing.
     * Modifies: weight
     * Effects:  returns current value of weight
     *           and resets weight to 0
     */
    int harvest();
private:
    // name of Vegetable, e.g., "Corn"
    string name;
    // amount of Vegetable, by weight
    int weight;
};
```

Q8.1. Which class members of an instance (i.e., variable) of the `Vegetable` data type, declared in the `main` function of a program, **can** be accessed using the dot operator `.`?

- ( ) `private`
- ( ) `public`
- ( ) `private` and `public`

Q8.2. Which of the following lines of code, in `main`, will create an instance of the `Vegetable` class with `name` of `"Corn"`?

```
A. Vegetable veggie;
B. Vegetable(string Corn);
C. Vegetable veggie("Corn");
D. Vegetable corn;
   corn.name = "Corn";
E. Vegetable "Corn";
```

- ( ) A.
- ( ) B.
- ( ) C.
- ( ) D.
- ( ) E.

Q8.3. Given the following `main` function, what is the correct way to change `peas` to be an instance of `Vegetable` with the `name` of `"snow peas"`?

```
int main() {
    Vegetable peas;
    // change the member variable 'name' of the
    // instance peas to "snow peas" here
    return 0;
}
A. peas.name = "snow peas";
B. peas.Vegetable("snow peas");
C. Vegetable peas("snow peas");
D. peas = Vegetable("snow peas");
```

- ( ) A.
- ( ) B.
- ( ) C.
- ( ) D.
- ( ) More than one of A, B, C, and D will change the `name` of the variable `peas` to `"snow peas"`.

Q8.4. Which of the following is a correct way to implement the member function `getName` just as it would appear in the file `Vegetable.cpp`?

```
A. Vegetable::getName() {
       return name;
   }
B. Vegetable::string getName() {
       return name;
   }
C. string Vegetable::getName();
D. string Vegetable::getName() {
       return Vegetable();
   }
E. string Vegetable::getName() {
       return name;
   }
```

- ( ) A.
- ( ) B.
- ( ) C.
- ( ) D.
- ( ) E.

Q8.5. Assume that `peas` is a previously declared instance of the `Vegetable` class. Which of the following is a valid call to the `grow` function?

```
A. cout << "Growing = " << peas.grow() << endl;
B. cout << "Growing = " << peas.grow(4) << endl;
C. peas.grow(4);
D. peas.grow();
```

- ( ) A.
- ( ) B.
- ( ) C.
- ( ) D.
- ( ) More than one of A, B, C, and D.

Q8.6. Which of the following declarations creates an array of `Vegetable` instances?

```
A. Vegetable veggies;
B. Vegetable veggies[18];
C. Vegetable veggies[3] = { Vegetable(), Vegetable("Corn"), Vegetable("Carrots") };
D. string Vegetable();
```

- ( ) A.
- (•) B.
- ( ) C.
- ( ) D.
- ( ) More than one of A, B, C, and D.

Q9. Complete the function definition below.

```
/**
 * Requires: ins stream in good state and a file has already
 *           been successfully opened
 * Modifies: ins, valNum, valStr
 * Effects:  Reads a number in format of numeric digits, e.g., 42
 *           If the operation fails, it will try to read a number
 *           in a format as a word, e.g., forty-two
 *
 *           If a number was read successfully as numeric digits, e.g., 42,
 *                 then set valNum to the value read
 *                 and set valStr to empty string, i.e., ""
 *           otherwise,
 *                 read number as a word and set valStr to the value read
 *                 and set valNum to 0
 *
 * Notes:    The word format will *never* contain spaces,
 *           e.g., "twenty one" will never be an input
 *           You should assume the file ends in a blank line
 *
 * Hint:     Take advantage of the fail-state flag in an input stream.
 *
 * Example:  An input of 9 should result in valNum = 9, valStr = ""
 *           An input of nine should result in valNum = 0, valStr = "nine"
 *
 */
void read(istream& ins, int& valNum, string& valStr) {
    // write your solution in the box below
    return;
}
```

Q10. Complete the function definition below.

```
/**
 * Requires: size >= 2 and size <= MAX_SIZE
 *           size is the number of rows and the number of columns for sheet
 * Modifies: sheet
 * Effects:  sets the first element of each row to the
 *           sum of all other elements in that row
 *
 * Example:  If sheet contains the following values for size == 3
 *           3  4  5
 *           8  11 10
 *           0  2  0
 *           then sheet would be modified to:
 *           9  4  5
 *           21 11 10
 *           2  2  0
 */
void sumRows(int sheet[MAX_SIZE][MAX_SIZE], int size) {
    // write your solution in the box below
    return;
}
```

Q11. The following class definitions for `Garden` (as it would appear in `Garden.h`) and `Vegetable` (as it would appear in `Vegetable.h`) are relevant to the remaining questions on the exam.

```
const int MAX_VEG = 10;
class Garden {
public:
    /**
     * Requires: Nothing.
```

```
     * Modifies: numVeg
     * Effects:  Default constructor. Sets numVeg to 0
     */
    Garden();
    /**
     * Requires: Nothing.
     * Modifies: numVeg, vegetables
     * Effects:  Adds the newVeg to the next available
     *           index of vegetables, increment numVeg,
     *           and return true
     *           If numVeg is already equal to MAX_VEG,
     *           the function should return false
     *           without making any changes.
     */
    bool plant(Vegetable newVeg);
    /**
     * Requires: Nothing.
     * Modifies: Nothing.
     * Effects:  Calls grow() for each element of the
     *           vegetables array for all Vegetable instances
     *           added by Garden::plant()
     *           i.e., within the bounds of numVeg
     */
    void grow();
    /**
     * Requires: Nothing.
     * Modifies: numVeg
     * Effects:  1. Calls harvest() for each element of the
     *              vegetables array for all Vegetable instances
     *              added by Garden::plant()
     *              i.e., within the bounds of numVeg.
     *           2. Sets numVeg to 0
     *           3. Returns total weight of all the vegetables harvested,
     *              i.e., the sum of the weights returned by
     *              Vegetable::harvest()
     */
        int harvest();
private:
    // the vegetables is an array of Vegetable
    Vegetable vegetables[MAX_VEG];
    // numVeg is the number of Vegetable class instances
    // previously added to vegetables array
    int numVeg;
};
class Vegetable {
public:
    /**
     * Requires: Nothing.
     * Modifies: name, weight
     * Effects:  Default constructor. Sets weight to 0,
     *           name to ""
     */
    Vegetable();
    /**
     * Requires: Nothing.
     * Modifies: name, weight
     * Effects:  Non-default constructor. Sets weight to 0,
     *           name to nameIn
     */
    Vegetable(string nameIn);
    /**
     * Requires: Nothing.
     * Modifies: Nothing.
     * Effects:  returns name
     */
    string getName();
    /**
     * Requires: Nothing.
     * Modifies: weight
     * Effects:  increment weight by 1
     */
    void grow();
    /**
     * Requires: Nothing.
```

```
     * Modifies: weight
     * Effects:  returns current value of weight
     *           and resets weight to 0
     */
    int harvest();
private:
    // name of Vegetable, e.g., "Corn"
    string name;
    // amount of Vegetable, by weight
    int weight;
};
```

Q11.1. Implement the `Garden` class member function `plant` below, **just as it would appear in the file** `Garden.cpp`. For your convenience, the function declaration from the `Garden` class definition is repeated below.

```
/**
 * Requires: Nothing.
 * Modifies: numVeg, vegetables
 * Effects:  Adds the newVeg to the next available
 *           index of vegetables, increment numVeg,
 *           and return true.
 *           If numVeg is already equal to MAX_VEG,
 *           the function should return false
 *           without making any changes.
*/
bool plant(Vegetable newVeg);
```

```
bool Garden::plant(Vegetable newVeg){
  if(numVeg < MAX_VEG){
    vegetables[numVeg] = newVeg;
    numVeg++;
    return true;
  }
  else{
    return false;
  }
}
```

Q11.2. Implement the `Garden` class member function `grow` below, **just as it would appear in the file** `Garden.cpp`. For your convenience, the function declaration from the `Garden` class definition is repeated below.

```
/**
 * Requires: Nothing.
 * Modifies: Nothing.
 * Effects:  Calls grow() for each element of the
 *           vegetables array for all Vegetable instances
 *           added by Garden::plant()
 *           i.e., within the bounds of numVeg
*/
void grow();
```

```
void Garden::grow(){
  for(int i=0; i<numVeg; i++){
    vegetables[i].grow();
  }
}
```

Q11.3. Implement the `Garden` class member function `harvest` below, **just as it would appear in the file** `Garden.cpp`. For your convenience, the function declaration from the `Garden` class definition is repeated below.

```
/**
 * Requires: Nothing.
 * Modifies: numVeg
 * Effects:  1. Calls harvest() for each element of the
 *              vegetables array for all Vegetable instances
 *              added by Garden::plant()
 *              i.e., within the bounds of numVeg.
 *           2. Sets numVeg to 0
 *           3. Returns total weight of all the vegetables harvested,
 *              i.e., the sum of the weights returned by
 *              Vegetable::harvest()
*/
int harvest();
```

```
int temp=0;
int Garden::harvest(){
  for(int i=0; i<numVeg; i++){
    temp += vegetables[i].harvest();
  }
  numVeg = 0;
  return temp;
}
```

Q11.4. Now, complete the gardening function below to use the Garden class.

```
void gardening() {
// declare a single Garden instance below
```

```
Garden gar;
```

```
// declare a Vegetable for planting in the garden below
// your Vegetable MUST have a name that is not an empty string
```

```
Vegetable veg("tomato");
```

```
/* plant the Vegetable in your garden below
 * if the Vegetable is not successfully planted:
 *      1. print "Cannot plant!"
 *      2. exit the gardening function
*/
```

```
int days = 0;
cout << "Enter number of days to grow (> 0) ";
cin >> days;
// now grow the garden days number of times
```

```
/* harvest the garden and print the total weight
 * example output:
 *     Vegetables harvested! Weight is 60
*/
cout << "Vegetables  harvested! Weight is ";
// add your code below
```

```
return;
}
```

```
|____|
```

```
/* plant the Vegetable in your garden below
 * if the Vegetable is not successfully planted:
 *      1. print "Cannot plant!"
 *      2. exit the gardening function
*/
```

```
|____|
```

```
int days = 0;
cout << "Enter number of days to grow (> 0) ";
cin >> days;
// now grow the garden days number of times
```

```
|____|
```

```
/* harvest the garden and print the total weight
 * example output:
 *     Vegetables harvested! Weight is 60
*/
cout << "Vegetables  harvested! Weight is ";
// add your code below
```

```
|____|
```

```
return;
}
```

```
|____|
```

```
/* plant the Vegetable in your garden below
 * if the Vegetable is not successfully planted:
 *      1. print "Cannot plant!"
 *      2. exit the gardening function
*/
```

```
|____|
```

```
int days = 0;
cout << "Enter number of days to grow (> 0) ";
cin >> days;
// now grow the garden days number of times
```

```
|____|
```

```
/* harvest the garden and print the total weight
 * example output:
 *     Vegetables harvested! Weight is 60
*/
cout << "Vegetables  harvested! Weight is ";
// add your code below
```

```
|____|
```

```
return;
}
```

```
|____|
```

```
/* plant the Vegetable in your garden below
 * if the Vegetable is not successfully planted:
 *      1. print "Cannot plant!"
 *      2. exit the gardening function
*/
```

```
|____|
```

```
int days = 0;
cout << "Enter number of days to grow (> 0) ";
cin >> days;
// now grow the garden days number of times
```

```
|____|
```

```
/* harvest the garden and print the total weight
 * example output:
 *     Vegetables harvested! Weight is 60
*/
cout << "Vegetables  harvested! Weight is ";
// add your code below
```

```
|____|
```

```
return;
}
```